Tomi Koskinen

PEREHDYTYSPROSESSIN DIGITALISOINTI WEB-SOVELLUKSEN AVULLA

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma
Toukokuu 2018

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Toukokuu 2018	**Tekijä/tekijät** Tomi Koskinen
Koulutusohjelma Tieto- ja viestintätekniikan koulutusohjelma		
Työn nimi PEREHDYTYSPROSESSIN DIGITALISOINTI WEB-SOVELLUKSEN AVULLA		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 29 + 1
Työelämäohjaaja Juho Muuraiskangas		

Opinnäytetyön aiheena oli digitalisoida ja automatisoida perehdytysprosessia tilaajayrityksessä. Perehdytysprosessin tueksi toteutettiin web-sovellus käyttöliittymineen ja sovelluslogiikoineen. Sovelluksen tarkoitus oli pitää kirjaa uudessa työpaikassa töiden aloittamiseen liittyvien tehtävien suorittamisesta ja auttaa prosessin koordinoinnissa useiden henkilöiden kesken. Toimeksiantajana oli Alfame Systems Oy, joka on Helsingissä ja Kokkolassa toimiva IT-alan yritys.

Sovellus toteutettiin käyttöliittymän osalta pääosin React-käyttöliittymäkirjaston avulla käyttäen valmiita React-Bootstrap –käyttöliittymäelementtejä. Perehdytykseen liittyvät tehtävät muotoiltiin liiketoimintaprosessikaavion muotoon. Activiti-prosessimoottori otettiin avuksi huolehtimaan kaaviossa kuvatun prosessin etenemisestä. Prosessimoottori ja käyttöliittymä yhdistettiin toimivaksi kokonaisuudeksi käyttämällä REST-rajapintaa.

Asiasanat
Activiti, liiketoimintaprosessi, JavaScript, React

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** May 2018	**Author** Tomi Koskinen
Degree programme Information Technology		
Name of thesis THE DIGITALIZATION OF THE INTRODUCTORY PROCESS WITH A WEB APPLICATION		
Instructor Sakari Männistö		**Pages** 29 + 1
Supervisor Juho Muuraiskangas		

The subject of this thesis was to digitalize and automatize the introductory process of the commissioning company. A web application was created including a user interface and an application layer to support the introductory period. The main purpose of the application was to help with the coordination of tasks between multiple persons and to keep track of the tasks that are expected to be completed during the introductory period. The commissioning company was Alfame Systems Oy, an IT company located in Helsinki and Kokkola.

The user interface of the application was mostly created with a user interface library called React and by utilizing ready-made React-Bootstrap user interface components. The tasks belonging to the introduction process were formalized as a business process diagram. The Activiti process engine was utilized to handle the execution of the process defined in the process diagram. The process engine and the user interface were connected by using a REST application programming interface.

Key words
Activiti, Business process, JavaScript, React

1 JOHDANTO

Opinnäytetyön aiheena on tutkia mahdollisuuksia digitalisoida ja automatisoida perehdytysprosessia. Pienessä yrityksessä henkilöstön perehdytykseen keskittynyttä osaamista ja resursseja on vähän, jolloin perehdytysprosessia automatisoimalla voidaan kehittää perehdytystä hyödyllisemmäksi ja tehokkaammaksi ja keskittää yrityksen resursseja ydinosaamiseen. Tavoitteena on opinnäytetyön puitteissa toteuttaa toimiva web-sovellus sisältäen käyttöliittymän ja sovelluslogiikan. Sovelluksen tarkoitus on toimia sekä perehdytettävän henkilön, että muiden perehdytykseen osallistuvien apuvälineenä. Tärkeimpänä ominaisuutena sovelluksen avulla pidetään kirjaa perehdytyksen etenemisestä. Lopputuloksena syntyä sovellus otetaan käyttöön tilaajayrityksessä uusien työntekijöiden ja sekä heidän perehdyttäjiensä avuksi. Toimeksiantajana on Alfame Systems Oy, joka on Helsingissä ja Kokkolassa toimiva integraatioihin ja digitalisaatioon keskittynyt IT-alan yritys.

Projektin kenties suurimpana haasteena on vähäinen kokemukseni sovelluskehityksestä ja etenkin kehitysprojektin työstämisestä tilaan, jossa se on otettavissa käyttöön. Käyttöliittymän ja sovelluslogiikan yhteensovittaminen on yksi haasteista. Käytettävistä tekniikoista Activiti-prosessimoottorista minulla on hieman aiempaa kokemusta, joten sen osalta perusasiat ovat hallinnassa. React-käyttöliittymäkirjastosta minulla ei ole aiempaa kokemusta, joten sen haltuunotto tiukassa aikataulussa on haastavaa.

Opinnäytetyöni tekemisen aikana tavoitteenani on oppia kehittämään toimiva käyttöliittymä sekä yhdistämään se rajapinnan avulla sovelluslogiikkaan. Etenkin toteutusvaiheessa React-käyttöliittymäkirjaston ja Activiti-prosessimoottorin teknisestä dokumentaatiosta oli apua. Lisäksi tavoitteenani on kehittyä tunnistamaan yrityksen liiketoimintaan liittyviä prosesseja ja oppia kehittämään ratkaisuja niiden automatisoimiseksi.

2 WEB-SOVELLUKSEN TOTEUTUKSEEN KÄYTETYT TEKNIIKAT

Web-sovelluksen rakenne voidaan yleensä jakaa kolmeen osaan: käyttöliittymään, sovelluslogiikkaan ja tietokantaan. Graafinen käyttöliittymä määrittelee, miltä sovellus näyttää käyttäjälle näyttöruudulla. Sovelluslogiikka huolehtii tiedon käsittelystä ja siirtämisestä sovelluksen osien välillä. Tietokantaan tallennetaan sovelluksen tarvitsemia tietoja. Tässä luvussa esittelen tärkeimmät tekniikat, joilla toteutin käyttöliittymä- ja sovelluslogiikkakerroksen. (Chang 2016.)

2.1 React

React on Facebookin kehittämä JavaScript-kirjasto. Reactin syntymää edelsi tarve löytää keinoja Cross Site Scripting -hyökkäyksien estämiseksi. Facebookin kehittämä ratkaisu ongelmaan oli uudenlainen versio PHP-ohjelmointikielestä nimeltään XHP. Se ei toisaalta auttanut toiseen ongelmaan, eli dynaamisten verkkosovellusten aiheuttamaan palvelinkuormitukseen. React kehitettiin vähentämään palvelimeen kohdistuvaa kuormaa siirtämällä prosessointia palvelimelta asiakkaan selaimeen. JavaScript-sisällön prosessointi on kuitenkin kuormittavaa, joten Reactin kehittämisessä keskityttiin yksinkertaistamaan web-sovellusten toteutusta. (Dawson 2014.)

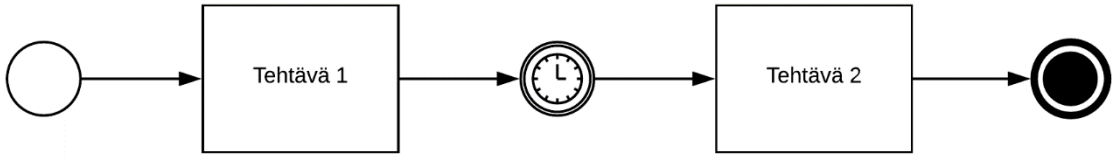
Reactin ydinajatus on, että koodilla kuvataan käyttöliittymän näkymä tietyllä ajanhetkellä, josta React muodostaa JavaScriptillä HTML-merkintäkielisen web-sivun. React-koodi koostuu komponenteista, joiden tila riippuu annetuista parametreistä ja tilamuuttujista. Aina kun komponentin tila muuttuu, käyttöliittymän näkymä päivitetään uudelleen. Jatkuva päivittäminen vaatisi paljon suoritustehoa, minkä vähentämiseksi Reactin ratkaisu on tilojen vertaaminen. Tietokoneen muistissa pidetään virtuaalista mallia edellisestä tilasta. Tilan muuttuessa React muodostaa JavaScriptillä uuden virtuaalisen mallin, vertaa vanhaa ja uutta mallia, ja sitten päivittää vain muuttuneet osat. Tämä vähentää muun muassa tarvittavaa laskentatehoa, sillä kaikkea sisältöä ei tarvitse piirtää näytölle uudelleen. (Buna 2017; Dawson 2014.)

2.2 Activiti ja liiketoimintaprosessien hallinta

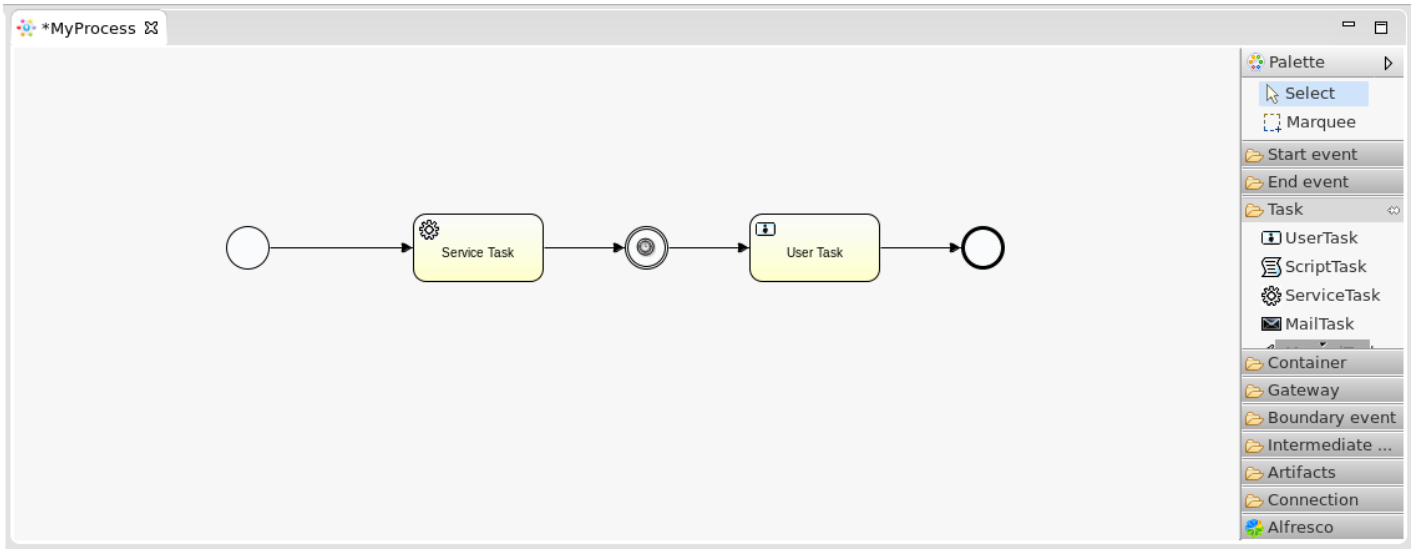
Liiketoimintaprosessi kuvaa toimintoja ja niiden suoritusjärjestystä kohti tavoitetta. Tavoite on tuottaa arvoa asiakkaalle tai toiselle liiketoimintaprosessille. Arvo taas on jotakin, jota asiakas tai jokin toinen liiketoimintaprosessi tarvitsee. Esimerkiksi jonkin organisaation koko ydintoiminta voidaan kuvata yhtenä korkean tason liiketoimintaprosessina, joka voidaan jakaa yksityiskohtaisemmin kuvattuihin aliprosesseihin. Aliprosessien avulla voidaan tarkastella tarkemmin liiketoimintaprosessin tietyn osa-alueen toimintaa. Kun prosessiin tehdään muutoksia, on tärkeää arvioida lopputulosta koko korkean tason prosessin kautta. Parannus johonkin toimintoon saattaa heikentää toista ja siten vaikuttaa jopa negatiivisesti lopputuotteena syntyvään arvoon. Liiketoimintaprosessien hallinnan tavoite on kehittää organisaation toimintaa siten, että organisaatio on tuottavimmillaan vallitsevissa olosuhteissa. (Kirchmer 2017, 3–5, 8.)

Activiti-projekti on Alfresco Software Incorporatedin rahoittama ja se on aloitettu vuonna 2010. Activiti on avoimen lähdekoodin prosessimoottori ja työnkulunhallinta-alusta. Prosessimoottorin tehtävä on tulkita ja suorittaa liiketoimintaprosessinhallintakaavioita. Liiketoimintaprosessien tavoitteena on kehittää ihmisten ja järjestelmien välistä toimintaa kohti liiketoiminnallista tavoitetta. Askeleet kohti tavoitetta voidaan kuvata kaaviomuodossa. Kaaviot voidaan jakaa abstrakteiksi ja suoritettaviksi tarkoitetuiksi kaavioiksi. Abstraktit kaaviot kuvaavat prosessin korkealla tasolla ja niitä suositellaan käytettäväksi ihmisten välisessä kommunikaatiossa. Suoritettavat kaaviot taas sisältävät ominaisuuksia, jotka ohjaavat prosessin suoritusta. (About Activiti.)

Activiti tukee Open Management Groupin vuonna 2004 kehittämää Business Process Management and Notationia (BPMN) ja Deciscion Making Notationia (DMN). Molemmilla standardeilla voidaan kuvata työnkulkuja tai liiketoimintaprosesseja. Työssäni käytin BPMN -muotoista prosessikaaviota. Prosessien kuvaamiseen käytetään vuokaavioita muistuttavia prosessikaavioita (KUVA 1). Kaaviot koostuvat erilaisista tehtävistä ja eri komponentteja yhdistävistä nuolista. Tehtävillä on yleensä tietty suoritusjärjestys, mutta prosessi voi vuokaavioiden tapaan myös haarautua, tai sisältää logiikkaa, joka ohjaa suoritusjärjestystä. Monimutkaisemmat prosessit voivat lisäksi sisältää aliprosesseja. Prosessikaavion sisältämät tehtävät voidaan luokitella käyttäjälle ja ohjelmistolle suunnattuihin tehtäviin suorittajan mukaan. Prosessikaavioita kuvataan XML-merkintäkielellä, mutta niitä voidaan luoda, muokata ja esittää myös graafisilla työkaluilla. (About Activiti; Mansuri & Laliwala 2014, 25–34.)



KUVA 1. Esimerkki BPMN-kaaviosta



KUVA 2. Liiketoimintaprosessikaavio muokattavana Activiti Designerissa

Activitissa on monia sovelluskehittäjälle hyödyllisiä ominaisuuksia. Esimerkiksi REST-rajapinta, jonka avulla pystytään monipuolisesti vaikuttamaan prosessin toimintaan. Prosesseja voidaan käynnistää ja niihin voidaan esimerkiksi lisätä liitetiedostoja tai tietoja prosessimuuttujiin. Lisäksi Activitissa on myös rajapinta, ns. Process Engine API, jota voidaan hyödyntää suoraan Java-kielisissä ohjelmissa. Activiti pitää kirjaa suoritettavista ja päättyneistä prosesseista, minkä avulla saadaan arvokasta tietoa prosessin suorituksesta, virhetilanteista ja tehokkuudesta. Activitin mukana tulee graafinen käyttöliittymä, jonka avulla onnistuu muun muassa prosessikaavioiden piirtäminen, prosessien käynnistäminen, käyttäjän toimintaa vaativien tehtävien suorittaminen sekä päättyneiden prosessien tutkiminen. (About Activiti; Laliwala & Mansuri 2014, 125–174, 51–98.)

3 LÄHTÖKOHTA JA ONGELMA

Rekrytointiprosessi käynnistyy henkilöstöjohtajan ja mahdollisen tulevan työntekijän yhteydenotolla. Yleensä melko aikaisessa rekrytointiprosessin vaiheessa tuore työntekijä saapuu ensimmäistä kertaa uudelle työpaikalleen ja aloittaa perehtymisen muun muassa työpaikan tapoihin ja käytäntöihin, työtehtäviinsä sekä tutustuu muuhun henkilöstöön. Yrityksen henkilöstöstä vastaavat olivat havainneet haasteita liittyen uusien henkilöiden perehdytysvaiheeseen. Haasteita tuottaa uuden henkilön perehdytyksen koordinointi sekä yrityksen pienehköstä koosta johtuen yksittäisen perehdytyksestä vastaavan henkilön puuttuminen.

Yrityksellä on käytössään itse tuotettua ohjeistusta perehdytykseen liittyvistä tehtävistä. Tehtäviä on jaoteltu sen mukaan, onko tehtävä tarkoitettu perehdytettävän omatoimisesti toteutettavaksi, vai esimerkiksi tuleviin tehtäviin valmentavaa koulutusta. Tehtävät pitävät sisällään käyttäjätunnusten luomista, sovellusten asentamista työssä käytettäviin laitteisiin, tutustumista yrityksen toimintaan, visioon ja arvoihin sekä muuta perehdytysvalmennusta. Aineistoa on melko runsaasti, mutta haasteena on sen esiintuonti ja tehokas käyttö perehdytyksen apuna.

Useimmat tehtävät tulevat suoritetuiksi ensimmäisten työviikkojen aikana, mutta tehtävien suoritusta ei välttämättä dokumentoida mitenkään. Tekemättömien tehtävien suorittaminen on siten perehdytettävän muistin varassa. Perehdytys jatkuu koeajan kuluessa ja kuukausien kuluttua on todennäköisesti vaikeuksia muistaa mitä asioita on vielä tekemättä. Lisäksi jo edellä mainittu prosessin tehtävien koordinointi aiheuttaa haasteita johtuen siitä, että perehdytysvastuuta on jaettu useille eri henkilöille. Heihin kuuluvat esimerkiksi sisäisistä järjestelmistä vastaavat henkilöt, luottamushenkilö, mentori ja henkilöstövastaava.

4 VAATIMUKSET

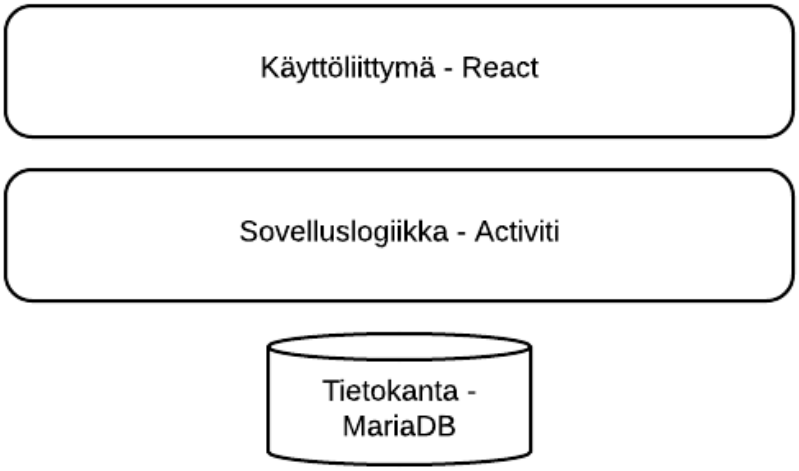
Ongelmakuvauksen pohjalta kartoitettiin aluksi korkean tason vaatimuksia sovellukselle. Ensimmäinen vaatimus on perehdytysprosessin tehtävien tilan dokumentointi. Se tarkoittaa sitä, että saatavilla täytyy olla tieto siitä, onko kyseinen tehtävä suoritettu vai vielä tekemättä. Tähän vaatimukseen liittyy myös seuraava vaatimus eli tiedon avoimuus: kerätty tieto pitäisi saada keskitettyä sijaintiin, josta sen voivat kaikki perehdytykseen liittyvät henkilöt tarvittaessa nähdä. Koska tiedon pitäisi olla mahdollisimman helposti saatavilla, yhdeksi vaatimukseksi nostettiin mahdollisuus käyttää sovellusta helposti useilla eri laitteilla, mutta ennen kaikkea älypuhelimella.

Määrityksen tarkentuessa päätettiin, ettei perehdytysprosessien historiasta pidetä kirjaa. Tausta-ajatuksena oli, että sovelluksen tarkoituksena on ennen kaikkea toimia apuna perehdytyksen aikana. Sen lisäksi prosessia on tarkoitus kehittää jatkuvasti paremmaksi, eikä vanhoihin käytäntöihin tarvitse palata. Lopulta viimeiseksi vaatimukseksi määritettiin jatkokehitysmahdollisuuksien huomioon ottaminen. Se tarkoittaa käytännössä sitä, että käytettävien työkalujen ja tekniikoiden tulisi olla yleisesti tunnettuja ja mielellään yrityksen sisällä käytettyjä. Toisaalta sovelluksesta tulisi tehdä sellainen, että uusien ominaisuuksien lisäämiseen tulisi varautua.

5 SOVELLUKSEN TOTEUTUS

Aiemmin määriteltiin, että tekniikoiden ja työkalujen tulisi olla sellaisia, että ne ovat tunnettuja ja niitä tuetaan jatkossakin. Web-sovellukseen käyttöliittymän luomiseen päätettiin käyttää React-kirjastoa. Tähän päädyttiin, koska Reactin kehityksen taustalla on iso organisaatio, eli Facebook, ja toisaalta yrityksen sisällä oli jo kokemusta sen käytöstä. Tällöin eteen tuleviin ongelmiin voi kysyä neuvoa ja oppaita sekä ratkaisuja ongelmiin löytyy Internetistä todennäköisemmin kuin tuntemattomamman tekniikan osalta. React-kehitykseen käytin tekstieditorina Visual Studio Codea, sillä minulla oli sen käytöstä aiempaa kokemusta.

Uuden työntekijän perehdytyksessä aiemmin käytetyn materiaalin sisältämät tehtävät voitiin hahmottaa prosessina. Ajatuksenani oli muodostaa perehdytyksen vaiheista prosessikaavio. Minulla oli hieman kokemusta Activiti-prosessimoottorista, joka voisi huolehtia prosessikaavion tulkinnasta ja suorituksesta. Tällä tavoin prosessia kuvaavaa kaaviota voitaisiin käyttää suoraan sovelluksen pohjana. Activitin rajapintoja hyödyntämällä on mahdollista yhdistää prosessimoottori ja web-sovellus. Prosessikaavion suunnitteluun käytin Activiti Designeria, joka toimii Eclipse-kehitysympäristön liitännäisenä.



KUVA 3. Sovelluksen kolmikerroksinen rakenne

Käytin toteutuksessa toimeksiantajan sisäisissä kehityshankkeissa käytettävää kehitysympäristöä. Kehitysympäristössä oli valmiiksi asennettuna Activiti, React-kehityksen aloittamiseen tarvittavat ohjelmistopaketit sekä tietokanta. Kehitysvaiheessa tietokantana käytettiin paikallisesti muistinvaraista H2-tietokantaa sekä testausta varten testipalvelinta, jossa on asennettuna MariaDB-tietokanta. Kuvassa 3 on esitetty sovellus kerroksittain ja kerroksia vastaavat työkalut.

Lisäksi otin käyttöön kehitysympäristöön asennetun npm-paketinhallinnan. Npm:ää käytetään web-kehityksessä projektin riippuvuuksien hallintaan. Riippuvuudet taas tarkoittavat projektin ulkopuolisia kirjastoja, tai moduuleja, joita kehitettävä sovellus vaatii toimiakseen. Projektille luodaan npm:ää käytettäessa package.json -niminen tiedosto, joka sisältää projektin perustiedot, muun muassa nimen, versionumeron ja lisenssin, sekä riippuvuudet versionumeroineen. Kehitys- ja julkaisuvaiheessa tarvittavat riippuvuudet on lueteltu tiedostossa erikseen, koska eri vaiheissa tarvitaan erilaisia työkaluja. Esimerkiksi kehitykseen käytettäviä testaustyökaluja ei tarvitse liittää julkaistavaan sovellukseen. Npm:ää käytetään komentoriviltä. Projekti aloitetaan komennolla ”npm init”, joka luo ohjatusti package.json -tiedoston. Moduulien asentaminen tapahtuu komennolla ”npm install” ja lisäämällä komennon jatkoksi asennettavan moduulin nimi. Moduuli lisätään riippuvuudeksi projektiin lisäämällä asennuskomennon loppuun vielä parametri ”-- save”. (Tierney 2017.)

5.1 Prosessikaavion suunnittelu

Liiketoimintaprosessikaavion pohjaksi päätettiin toimeksiantajan kanssa ottaa perehdytysprosessin apuna aiemmin ollut tehtävälista. Aloin suunnitella kaaviota Activiti Designerilla ja listan pohjalta kehitinkin muutamia erilaisia luonnoksia kaaviosta. Parhaiten projektin vaatimuksia vastaava versio kaaviosta sisältää kaikki perehdytykseen liittyvät tehtävät omina itsenäisinä tehtävinään. Tehtävistä salasanasovellus- ja sähköpostitilin luonti ovat vaatimuksena muiden käyttäjätilien luomiselle. Siksi sijoitin pakolliset ja tärkeimmät tehtävät prosessin alkuun. Muiden tehtävien kohdalla suoritusjärjestyksellä ei ole merkitystä. Kun kaikki tehtävät on suoritettu, prosessi päättyy loppupisteeseen.

Myöhemmin käyttöliittymää toteuttaessani, päätimme toimeksiantajan kanssa tehdä muutoksia kaavioon (LIITE 1). Tehtäviin lisättiin tarkempia lisätietoja ja ohjeita, jotka auttavat tehtävän suorituksessa. Lisätietojen tallentamiseen käytetään liiketoimintaprosessin käyttäjätehtävään sisältyvää kuvaustekstikenttää. Lisäksi samantyyppisiä tehtäviä ryhmiteltiin kaaviossa, esimerkiksi tarvittavien ohjelmistojen asennukseen liittyvät tehtävät ovat vierekkäin. Tällöin kaaviota muokatessa tai käyttöliittymän kautta tarkasteltaessa tehtävät löytyvät todennäköisesti helpommin.

5.2 Käyttöliittymän suunnittelu

Liiketoimintaprosessikaavion suunnittelun jälkeen aloin suunnitella käyttöliittymää. Suunnittelu tapahtui aluksi paperille luonnostellen. Tässä vaiheessa päädyin pitkälti lopullisen kaltaiseen ulkoasuun. Sen jälkeen tein Reactilla luonnosten pohjalta muutamia luonnosmaisia versioita ilman toiminnallisuutta. Tässä vaiheessa esiin nousi tarve löytää keino sisällön asetteluun eri tavoin riippuen näyttöruudun koosta. Lisäksi tarvitsin selkeitä käyttöliittymäelementtejä parantamaan sovelluksen käytettävyyttä ja ulkonäköä.

Löysin ratkaisun sekä sisällön asettelun haasteisiin, että käytettävyyden parantamiseen Bootstrap-kirjastosta. Se sisältää valmiiksi määriteltyjä käyttöliittymäkomponentteja, muun muassa painikkeita, lomakkeita ja navigointipalkkeja. Lisäksi Bootstrapiin sisältyy komponentteja, jotka auttavat sisällön asettelun hienosäätämisessä. Bootstrap ei ole suoraan yhteensopiva Reactin kanssa, joten otin käyttöön React-Bootstrap-käyttöliittymäkirjaston lisäämällä sen projektin riippuvuudeksi npm-paketinhallinnan avulla. React-Bootstrap-kirjasto sisältää Bootstrap-komponentteja uudelleenkirjoitettuna React-komponenteiksi. Komponenttien ulkonäköä voi muokata kirjoittamalla korvaava CSS-tyylitiedosto komponentille tai käyttämällä valmiita teemoja. (Getting started – Introduction; Ludosky 2015.)

React-Bootstrap tarjoaa helpohkon tavan sivuston asettelun muodostamiseen. Käytin Col- ja Row -elementtejä (KUVA 4), jotka mahdollistavat sisällön skaalautumisen niin puhelimen pienille, kuin pöytäkoneen isommille näytöille. Skaalautumista eri näyttökokoja varten voi hallita Col-elementin attribuuttien avulla. Elementtien muodostaman ruudukon avulla on mahdollista keskittää sisältö suurilla näytöillä, jolloin sisällön leveys ei kasva liian suureksi. Pienemmillä näytöillä taas saadaan hyödynnettyä laitteen kuvapinta-ala mahdollisimman laajalti. (Layout – Grid system.)

```
<div>
    <Grid>
        <Row>
            <Col md={8} mdOffset={2}>
                ...
            </Col>
        </Row>
    </Grid>
</div>
```

KUVA 4. Koodiesimerkki 1

Staattisen luonnoksen tekemisen jälkeen aloin harjoitella sisällön muuttamista Reactilla luotuihin näkymiin. Otin kehitykseen mukaan tässä vaiheessa myös ensimmäiset kutsut rajapintaan.

5.3 Activitin rajapinnan kutsuminen

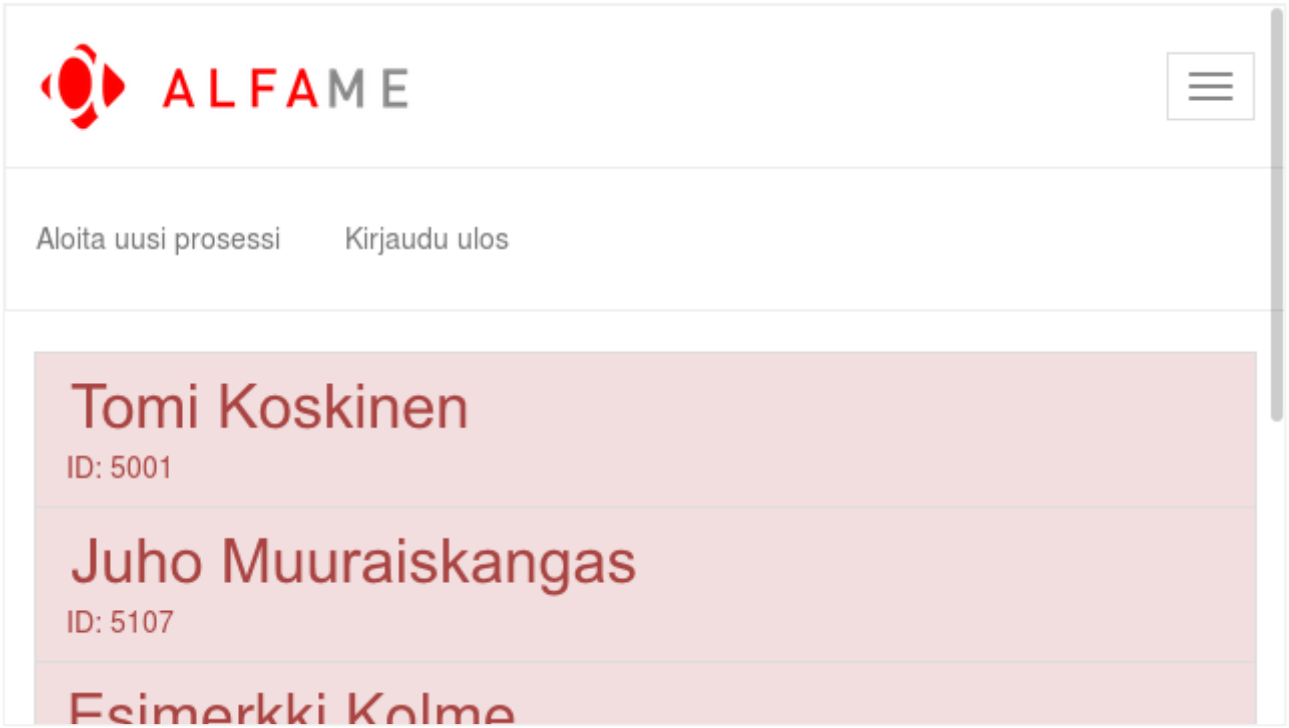
Aluksi täytyi tutustua Activitin REST-rajapintaan ja tutkia sen toimintaa dokumentaation ja RESTClient -selainliitännäisen avulla. Prosessi-instansseja voidaan käynnistää, ja niiden käyttäjille suunnattujen tehtävien tilaan voidaan vaikuttaa, lähettämällä HTTP-kutsuja, kuten GET ja POST, rajapinnan tarjoamiin päätepisteisiin. Rajapinta on saatavissa osoitteessa ”activiti-webapp-rest2-5.22.0/service”, jonka jatkoksi liitetään päätepiste riippuen siitä, mikä toiminto halutaan suorittaa. Esimerkiksi aktiiviset prosessi-instanssit voidaan noutaa paikallisessa ympäristössä osoitteesta ”https://localhost/activiti-webapp-rest2-5.22.0/service/runtime/process-instances”. Päätepistettä kutsuessa osoitteeseen voidaan myös lisätä parametrejä, joiden avulla saadaan rajattua tuloksia esimerkiksi tietyn prosessi-instanssin tai tehtävän tunnisteen mukaan.

Kutsujen tekemiseen otin käyttöön jQuery-kirjaston ajax-funktion, koska minulla oli siitä aiempaa kokemusta. JQuery on JavaScript -kirjasto, jonka tehtävä on helpottaa ja nopeuttaa alustariippumattoman koodin kirjoittamista, sisällön manipulointia ja animointia. Näkyvänä erona puhtaaseen JavaScript-koodiin jQuery-funktiot alkavat dollarimerkillä. Ajax on web-kehitystapa, jonka tarkoituksena on viestiä palvelimen ja käyttäjän selaimen välillä asynkronisesti. Kutsut palvelimelle ja vastausten käsittely tapahtuvat taustalla, jolloin käyttäjän vuorovaikutus sivuston tai sovelluksen kanssa ei keskeydy. JQuery sisältää valmiita funktioita palvelinkutsujen tekemiseen asynkronisesti. (Doyle 2012; Rascia 2017; Tero 2014.)

5.4 Navigaatio ja uuden prosessin aloittaminen

Sovelluksen päänäkymä, eli henkilönäkymä, kokoaa käynnissä olevat perehdytysprosessit yhteen listaukseen. Näytön yläreunan otsikkopalkki on kiinteästi näkyvillä. Nykypäivänä verkkopalveluissa yleinen tapa on mahdollistaa etusivulle palaaminen palvelun tai yrityksen logoa painamalla. Tässä tapauksessa otsikkopalkin logo johdattaa käyttäjän muista näkymistä henkilönäkymään. Tämä ominaisuus pa-

rantaa käytettävyyttä, sillä käyttäjällä on aina näkyvillä mahdollisuus palata päänäkymään. Lisäksi ominaisuus on tuttu muista verkkopalveluista, mikä lisää uuden käyttäjän tuttuudentunnetta. (Krug 2014, 62.)



KUVA 5. Henkilönäkymä, jossa valikko avoinna

Otsikkopalkissa on myös valikko, jonka toimintoihin on tarpeellista päästä käsiksi mistä tahansa sovelluksen näkymästä. Näihin toimintoihin kuuluvat uuden prosessin aloittaminen ja uloskirjautuminen. Kirjautumisen toteutus jäi opinnäytetyön aihealueen ulkopuolelle. Valikon kuvakkeena käytetään nykyään muun muassa Googlen verkkopalveluista ja Android-käyttöjärjestelmästä tuttua hampurilaisvalikkokuvaketta. Valikko avautuu otsikkopalkin alapuolelle (KUVA 5). Suurilla näyttöpinta-aloilla valikon toiminnot näkyvät suoraan otsikkopalkissa.

Käynnistettävä prosessi valitaan sisällyttämällä rajapintakutsuun tieto prosessin yksilöllisestä tunnisteesta. Tunniste koostuu tekstimuotoisesta prosessiavaimesta, joka annetaan liiketoimintaprosessikaaviota luotaessa, sekä versionumerosta, jota Activiti kasvattaa aina kaaviota päivitettäessä. Kun käyttäjän käynnistää prosessin, noudetaan aluksi tunnetun prosessiavaimen perusteella prosessin versiot. Jos prosessista on useampia versioita, valitaan versionumeroltaan suurin, eli uusin, prosessitunniste (KUVA 6).

```
getProcessDefinitionId(){

    let processDefinitionKey  = "robottimanageri_newPerson";
    let processDefinitionId   = '';
    let definitionVersion     = 0;
    let latestDeploymentId    = 0;
    let latestDefinitionIndex = 0;

    $.ajax({
    type: "GET",
    url: `${root.config.apiBaseUrl}/repository/process-definitions?key=${processDefinitionKey}&size=50`,
    contentType: "application/json"
    }).then(( response ) => {

        console.log( "Search for newest deployment");
        for( var i = 0; i < response.data.length; i++ ){
            console.log(parseInt(response.data[i].deploymentId) + " > " + parseInt(latestDeploymentId));
            if( parseInt( response.data[i].deploymentId ) > parseInt( latestDeploymentId ) ){
                definitionVersion = response.data[i].version;
                latestDeploymentId = response.data[i].deploymentId;
                console.log("Found newer definition " + definitionVersion + ":" + latestDeploymentId);
                latestDefinitionIndex = i;
            }
            else{
                console.log("Older version, skipping: " + response.data[i].version + ":" + response.data[i].deploymentId);
            }
        }
        console.log( "definitionVersion: " + definitionVersion + ", latestDeployId: " + latestDeploymentId);

        processDefinitionId = response.data[latestDefinitionIndex].id;
        console.log( "procDefId: " + processDefinitionId );

        this.startProcess( processDefinitionId );
    }), ()=>{
        console.log( "No process definition found!" );
    };
}
```

KUVA 6. Koodiesimerkki 2

Kun uusin tunniste on löydetty, prosessin suoritus käynnistetään lähettämällä POST-kutsu rajapintaan. Kutsuun sisällytetään prosessin tunniste JSON-muodossa dataosaan. Rajapinta lähettää kuittauksen mukana käynnistetyn prosessi-instanssin tunnisteen, joka tallennetaan tilamuuttujaan (KUVA 7).

```
startProcess( procDefId ){
    let processDefinitionId = procDefId;
    let processInstanceId   = '';

    $.ajax({
    type: "POST",
    url: `${root.config.apiBaseUrl}/runtime/process-instances`,
    contentType: "application/json",
    data: `{"processDefinitionId": "${processDefinitionId}"}`
    })
    .then( ( response ) => {

        console.log("startProcess " + response.id);
        processInstanceId = response.id;
        this.setState( { processInstanceId : processInstanceId } );
        this.createNewPersonVariables();
    }), () => {
        console.log( "Process not started!" );
    };
}
```

KUVA 7. Koodiesimerkki 3

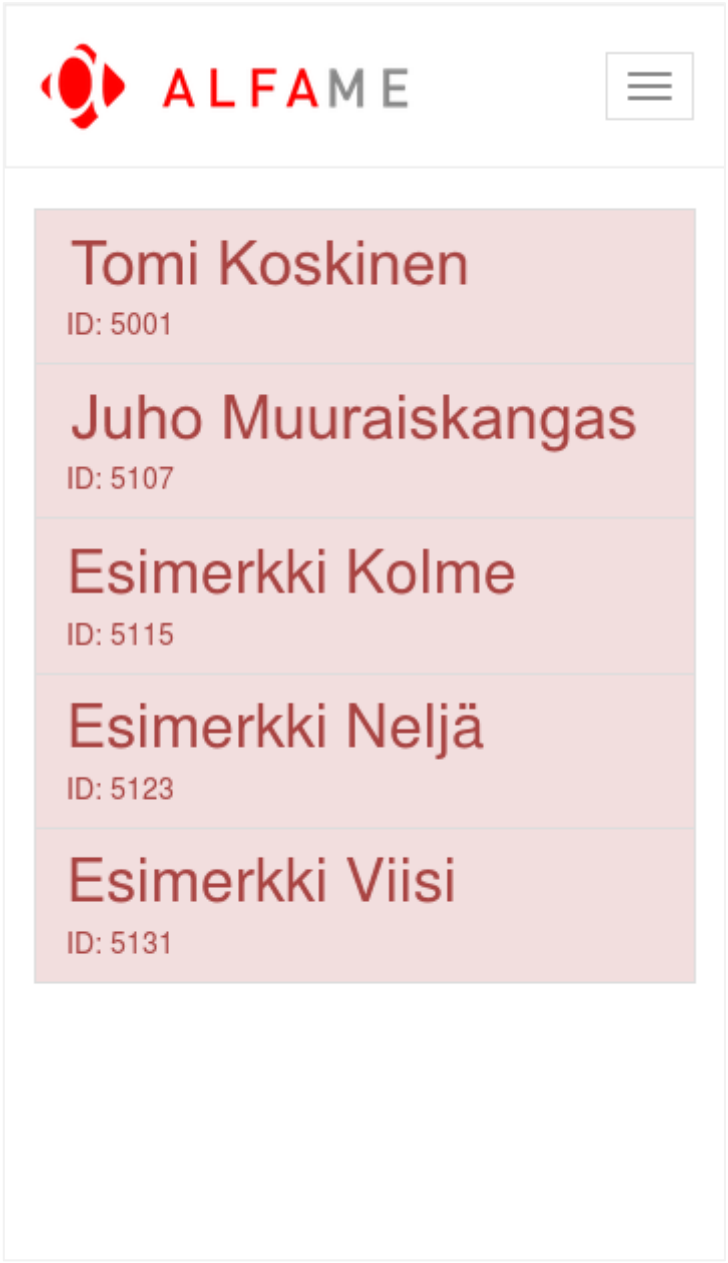
Prosessi-instanssiin sisällytetään ennen prosessin käynnistämistä syötetyt tiedot, nimi ja sähköpostiosoite, käyttämällä hyväksi prosessin aloituskutsun vastauksessa saatua prosessi-instanssin tunnistetta. Henkilön tiedoista tehdään JSON-taulukko, jossa annetaan prosessimuuttujan nimi, tyyppi ja arvo. Arvoiksi annetaan siis henkilön nimi ja sähköpostiosoite. Kun rajapinta lähettää vastauksen, suljetaan prosessin aloitusikkuna ja siirrytään HashHistory-kirjaston avulla prosessi-instanssin tehtävänäkymään (KUVA 8).

```
createNewPersonVariables(){

    let name              = this.state.newPersonName;
    let email             = this.state.newPersonEmail;
    let processInstanceId = this.state.processInstanceId;

    let processVariables = [
                            {
                                "name":"newPersonName",
                                "type":"string",
                                "value":`${name}`
                            },
                            {
                                "name":"newPersonEmail",
                                "type":"string",
                                "value":`${email}`
                            }];
    $.ajax( {
        type: "POST",
        url: `${root.config.apiBaseUrl}/runtime/process-instances/${processInstanceId}/variables`,
        contentType: "application/json",
        data: JSON.stringify( processVariables )
        })
        .then( ( response ) => {

            this.setState( { newProcessDialogOpen : false } );
            hashHistory.replace('/task');
            hashHistory.push(`/task/${processInstanceId}`);
        });
}
```

KUVA 8. Koodiesimerkki 4

5.5 Henkilönäkymä

Uusien henkilöiden perehdytysprosessit on sijoitettu näkyvälle paikalle keskelle näyttöä (KUVA 9). Prosesseja kuvaavista Panel-komponenteista tehtiin riittävän suuria, jotta niiden valitseminen olisi helppoa myös puhelimella. Käyttäjän saapuessa henkilönäkymään, sovellus noutaa käynnissä olevien perehdytysprosessien tunnisteet prosessimoottorin rajapinnasta. Sitten sovellus luo jokaiselle prosessille oman elementin, asettaa yksilöllisen tunnisteen elementin avaimeksi ja henkilön nimen elementin otsikoksi.



KUVA 9. Henkilönäkymä älypuhelimen näytöllä

Käyttäjän saapuessa sovellukseen, avautuu ensimmäisenä henkilönäkymä. React-komponenttia luotaessa suoritetaan componentWillMount()-metodi, jonka avulla haetaan käynnissä olevat prosessi-instanssit (KUVA 10). Vastauksena saatujen prosessi-instanssien tunnisteet tallennetaan tilamuuttujaksi. Tunnisteiden avulla tehdään myös jokaiselle prosessi-instanssille kutsu, jolla noudetaan prosessiin liitetyn henkilön nimi (KUVA 11). (React.Component.)

```
componentWillMount(){

    let processDefinitionKey = "robottimanageri_newPerson";
    this.listProcessInstances( processDefinitionKey );
}
```

KUVA 10. Koodiesimerkki 5

```
parseProcessIds( response ){

    let processIds = [];

    for( var i = 0; i < response.data.length; i++ ){
        processIds.push( response.data[i].id );
    }

    this.setState( { tempProcessIds : processIds } );
    this.getProcessVariable( processIds, 0, "newPersonName" );

}
```

KUVA 11. Koodiesimerkki 6

Ennen kuin henkilöiden nimiä aletaan noutaa, luodaan taulu, joka sisältää JSON-objekteja. Jokainen objekti sisältää prosessi-instanssin tunnisteen ja henkilön nimelle tarkoitetun kentän. Henkilön nimiä noudettaessa getProcessVariable-metodi kutsuu itseään silmukassa, kunnes kaikki prosessi-instanssit on käyty läpi. Tämä täytyy tehdä siksi, että muuttujia noudetaan prosessi-instanssin tunnisteen perusteella yksitellen. Rajapinnan palauttaessa nimimuuttujan, se lisätään aiemmin luotuun tauluun samaan objektiin muuttujan sisältävän prosessi-instanssin tunnisteen kanssa. Mikäli prosessi-instanssilla ei ole henkilön nimeä sisältävää muuttujaa, kyseinen instanssi ohitetaan ja jatketaan jäljellä olevien muuttujien noutamista. Lopuksi prosessi-instanssien tunnisteet ja henkilöiden nimet sisältävä taulu asetetaan tilamuuttujaan, jolloin näkymä päivittyy (KUVA 12).

```
getProcessVariable( procInstIds, index, variableName ){
    let instanceIds      = procInstIds;
    let processIds       = this.state.tempProcessIds;
    let runningProcesses = [];
    let obj              = {};

    if( index == 0 ){
        for( var i = 0; i < instanceIds.length; i++ ){
            obj = { "processId" : instanceIds[i], "processName" : "uusi henkilö" };
            runningProcesses.push( obj );
        }
        this.setState( { tempNames : runningProcesses } );
    }
    else{
        runningProcesses = this.state.tempNames;
    }
    let processInstanceId = instanceIds[ index ];
    let variable = variableName;
    $.ajax({
    type: "GET",
    url: `${root.config.apiBaseUrl}/runtime/process-instances/${processInstanceId}/variables/${variable}`,
    contentType: "application/json",
    }).then( ( response ) => {
        if( response.value ){
            runningProcesses[index].processName = response.value;
            this.setState( { tempNames : runningProcesses} );
        }
        let newIndex = index + 1;
        if( newIndex >= instanceIds.length ){
            this.setState( { runningProcesses : runningProcesses } );
        }
        else{
            this.getProcessVariable( instanceIds, newIndex, "newPersonName" );
        }
    }, () => {
        let newIndex = index + 1;
        if( newIndex >= instanceIds.length ){
            this.setState( { runningProcesses : runningProcesses } );
        }
        else{
            this.getProcessVariable( instanceIds, newIndex, "newPersonName" );
        }
    });
}
```

KUVA 12. Koodiesimerkki 7

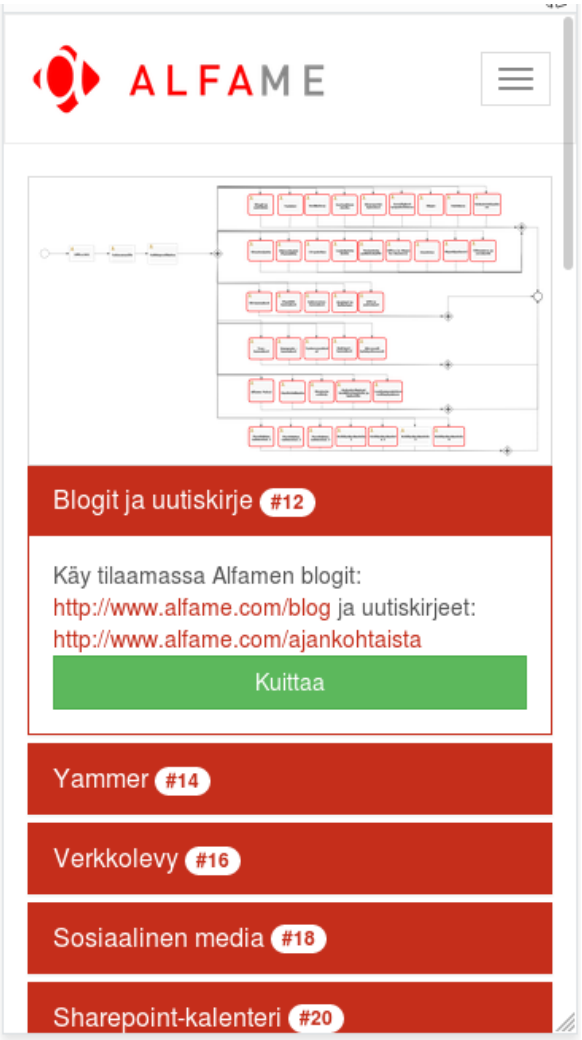
Sovelluksessa näkymästä toiseen siirtymiseen käytetään React Router -kirjastoa, joka osaa osoitteen avulla kertoa käyttäjän sijainnin näkymähierarkiassa. React Routerin hashHistory-implementaation avulla voidaan myös manipuloida käyttäjän sijaintia. Käyttäjä ohjataan osoitteeseen /task/${processId}, jossa sijaitsee tehtävänäkymä. ProcessId-muuttuja on yhdistetty käyttäjän valitsemaan painikkeeseen. btnClicked-tilan muutos aiheuttaa sivun päivittymisen, jolloin näytetään tehtävänäkymä. Henkilön nimeä painamalla suoritetaan kuvan 13 mukainen funktio. (Dabit 2016.)

```
onButtonClick( processId ){

    hashHistory.push( `/task/${processId}` );
    this.setState( { btnClicked : true } );
}
```

KUVA 13. Koodiesimerkki 8

5.6 Tehtävänäkymä

Tehtävänäkymässä liiketoimintaprosessikaavio on keskeisellä paikalla, otsikkopalkin alapuolella (KUVA 14). Kaavio antaa nopeasti käyttäjälle yleisvaikutelman siitä, missä vaiheessa prosessi on. Prosessimoottori merkitsee käyttäjän toimintaa odottavat tehtävät punaisella ääriviivalla. Siten voidaan nopeasti nähdä, mitkä tehtävät vaativat käyttäjän huomiota. Suoritetuista tehtävistä ääriviiva poistuu. Kaavion alapuolella sijaitsevat suorittamattomat tehtävät. Jokaisella kaavioon piirretyllä ja määritellyllä tehtävällä on oma paneeli, jonka otsikkoa painamalla paneeli laajenee ja näyttää tehtävän lisätiedot. Ikkunassa on tehtävän tarkempi kuvaus, sekä painike, josta voi kuitata tehtävän tehdyksi. Kun kaikki tehtävät on suoritettu, määriteltiin, että viimeisen tehtävän kuittaamisen jälkeen näytölle jää ainoastaan kuva kaaviosta, jossa kaikki tehtävät on suoritettu. Tehtävänäkymästä poistumisen jälkeen ei päättyneeseen prosessiin enää voi palata, sillä prosessimoottorin rajapinnasta noudetaan vain aktiiviset prosessi-instanssit.



KUVA 14. Tehtävänäkymä

Kaavio on toteutettu Image-komponenttina, johon kuva tuodaan tilamuuttujasta (KUVA 15). Tilamuuttujaan kuva noudetaan prosessimoottorin rajapinnasta käyttäjän saapuessa tehtävänäkymään. Attribuutti ”responsive” saa kuvan mukautumaan näkymän asetteluun ja ”thumbnail” erottaa kuvan taustasta ohuella reunaviivalla. (Components – Images.)

```
<Image src={ this.state.processDiagram } responsive thumbnail/>
```

KUVA 15. Koodiesimerkki 9

Kuten henkilönäkymässäkin, tehtävänäkymään siirryttäessä, tai sen päivittyessä React suorittaa componentWillMount()-funktion. Funktiossa käynnistetään prosessikaavion noutaminen (KUVA 16). Prosessikaavio vastaanotetaan 16-bitin pituisina merkkeinä, eli sanoina, ja se täytyy muuntaa ensin tavuiksi tekemällä AND-operaatio ”& 255” ja sitten base64-enkoodata merkkijono selaimen ymmärtämäksi binääriksi. Kuvaan lisätään myös tieto tiedostomuodosta. Sen jälkeen kuva tallennetaan tilamuuttujaan käyttöliittymän käytettäväksi. (Tenenhaus 2010.)

```
getProcessDiagram( procInstId ){
    let processInstanceId = procInstId;

    $.ajax( {
        url: `${root.config.apiBaseUrl}/runtime/process-instances/${processInstanceId}/diagram`,
        method: 'get',
        contentType: 'application/json',
        processData: false,
        beforeSend: ( xhr ) => {
                xhr.overrideMimeType( 'text/plain; charset=x-user-defined' );
        }
    }).then( ( diagramResponse ) => {

        let binary = '';

        for( let j = 0; j < diagramResponse.length; j++ ) {
                binary += String.fromCharCode( diagramResponse.charCodeAt(j) & 255 );
        }

        let diagram = '' + window.btoa( binary );
        this.setState( { processDiagram : diagram } );

        this.listTasksByProcess( processInstanceId );
    }, () => {
        console.log( "No diagram returned" );
        this.listTasksByProcess( processInstanceId );
    });
}
```

KUVA 16. Koodiesimerkki 10

Prosessikaavion hakemisen jälkeen noudetaan vielä prosessi-instanssille kuuluvat tehtävät, mikä tapahtuu pitkälti samalla tavoin kuin prosessi-instanssienkin noutaminen. Kutsu osoitetaan tasks-päätepisteelle ja parametriksi lisätään prosessi-instanssin tunniste (KUVA 17).

```
listTasksByProcess( procInstId ){

    let processInstanceId = procInstId;

    $.ajax({
    type: "GET",
    url: `${root.config.apiBaseUrl}/runtime/tasks?processInstanceId=${processInstanceId}&size=50`,
    contentType: "application/json",
    }).then( ( response ) => {


        if(response.data.length > 0){

            let taskList = response;
            this.parseTaskData(taskList);
        } else {
            console.log("No tasks returned!");
        }
    }, () => {
        console.log("No tasks!");
    });
}
```

KUVA 17. Koodiesimerkki 11

Vastaanotettujen tehtävien tiedoista koostetaan vielä JSON-muotoinen objekti, joka sisältää tehtävän tunnisteen, nimen ja kuvauksen. Lopuksi tehtävien tiedot tallennetaan tilamuuttujaan. Tilamuuttujaan tallentaminen aiheuttaa jälleen näkymän päivittymisen, jolloin onnistuneesti noudetut tehtävät ja kaavio tulevat näkyviin (KUVA 18).

```
parseTaskData( response ){

    let taskData = response;
    let taskList = [];
    let task     = {};


    for(var i = 0; i < taskData.data.length; i++){

        task = { "taskId" : taskData.data[i].id,
                 "taskName" : taskData.data[i].name,
                 "taskDescription" : taskData.data[i].description };
        taskList.push( task );

    }


    this.setState( { runningTasks: taskList } );
}
```

KUVA 18. Koodiesimerkki 12

Jokaisella tehtävällä on oma Panel-komponentti. RunningTasks-tilamuuttuja sisältää JSON-objekteina prosessi-instanssin suorittamattomat tehtävät. Panelit luodaan runningTasks-tilamuuttujan pohjalta map-funktiolla. Siten jokaiselle Panel-komponetille saadaan asetettua tehtävän tunniste, nimi ja lisätiedot. Paneleilla on collapsible-attribuutti, joka mahdollistaa Panelin laajentamisen ja pienentämisen. Panelit ovat PanelGroup-komponentin sisällä (KUVA 19), jolloin PanelGroupin accordion-attribuutilla voidaan määritellä, että Paneleita on laajennettuna vain yksi kerrallaan. PanelGroupin activeKey-attribuutti käyttää tilamuuttujaa, joka asetetaan Panelia painettaessa handlePanelClick-funktion avulla. Siten PanelGroup tietää, mikä Paneleista on aktiivinen. Paneleiden sisältö, eli tehtävän lisätiedot, on Linkify-tagien sisällä, jolloin Linkify-JavaScript-kirjasto muuntaa web-osoitteet hyperlinkeiksi. Lisäksi Panel sisältää Button-komponentin, eli painikkeen, jolla voi kuitata tehtävän. Painikkeen block-attribuutti keskittää tekstin. (Components – Panels; Linkify-react.)

```
<PanelGroup activeKey={this.state.activePanelKey} accordion>
    { this.state.runningTasks.map( ( task, i )=>{
    return( <Panel onClick={ () => {this.handlePanelClick( task.taskId )}}
                   eventKey={task.taskId}
                   header={<p>{task.taskName.toString()} <Badge>#{task.taskId}</Badge></p>}
                   bsStyle="primary" key={task.taskId} collapsible>
                    <Linkify>{ task.taskDescription }</Linkify>
                    <Button bsStyle="success"
                            onClick={ () => {this.checkoutTask( task.taskId )}} block>
                                Kuittaa
                    </Button>
                </Panel>);
            })}
</PanelGroup>
```

KUVA 19. Koodiesimerkki 13

Kun tehtäväikkunan kuittauspainiketta painetaan, lähetetään POST-kutsu rajapintaan. Kutsuun liitetään JSON-objekti, joka sisältää halutun toiminnon sekä haluttaessa valinnaiset muuttujat ja kutsu ohjataan runtime/tasks-päätepisteeseen. Kuittauksen onnistuttua noudetaan uudelleen prosessikaavio ja suorittamatta olevat tehtävät. Näin saadaan näkyviin päivittynyt kaaviokuva ja kuitattu tehtävä poistuu näkyvistä (KUVA 20).

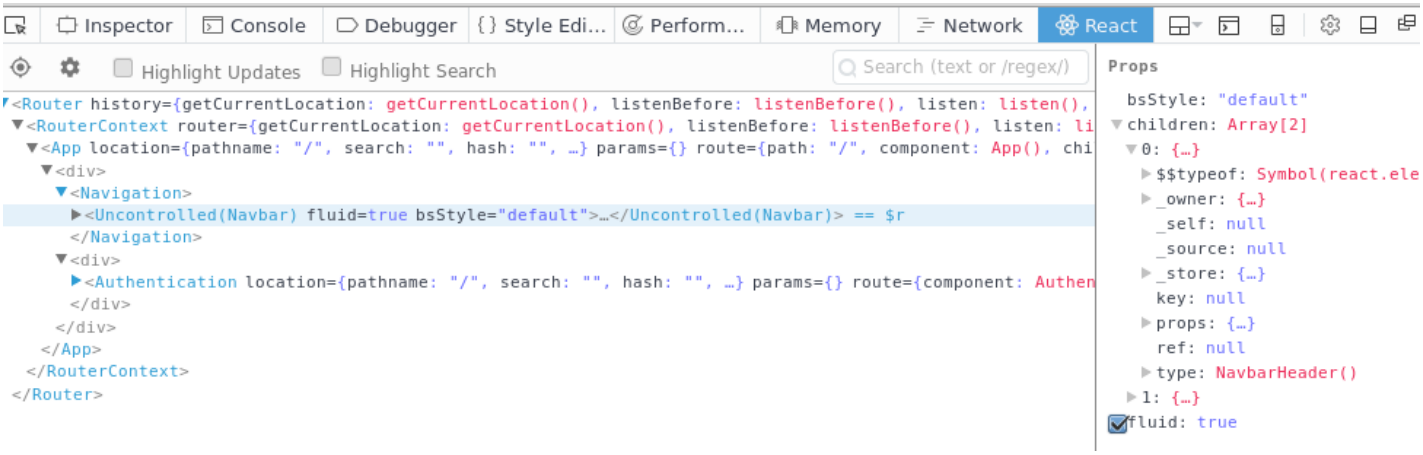
```
completeTask( taskId ){

    let username = User.getUsername();
    let requestVariables = {"action":"complete",
                                "variables":[{
                                    "name":"notes",
                                    "value":`Task ${taskId} done by ${username}`,
                                    "type":"string"
                                    }]};

    $.ajax({
    type: "POST",
    url: `${root.config.apiBaseUrl}/runtime/tasks/${taskId}`,
    contentType: "application/json",
    data: JSON.stringify(requestVariables),
    }).then( ( response ) => {


        this.setState( { runningTasks: [] } );
        let processId = this.state.processId;
        this.getProcessDiagram( processId );
    });
}
```

KUVA 20. Koodiesimerkki 14

6 TYÖNKULKU JA TESTAUS

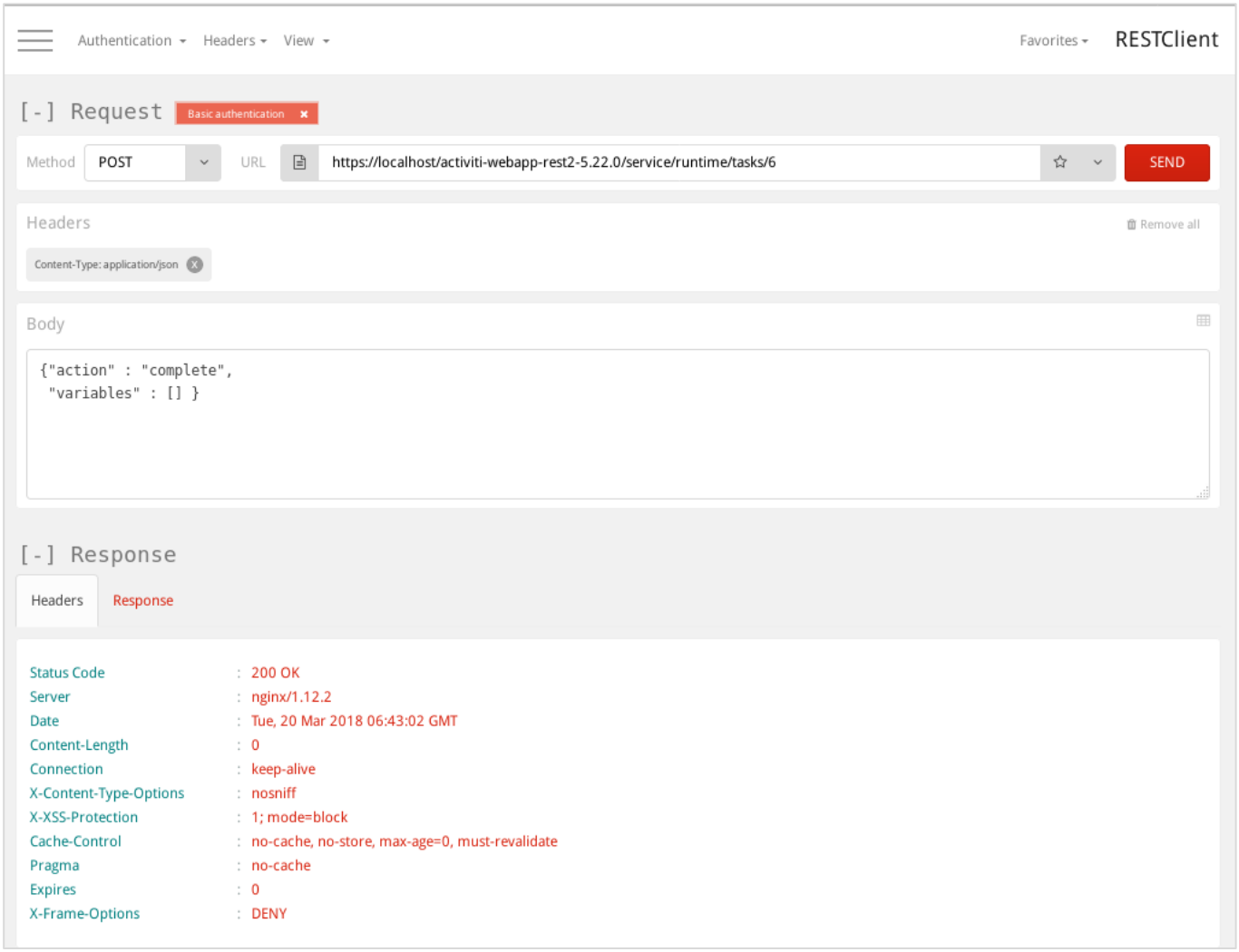
Sovellusta kehittäessä pyrin lisäämään ominaisuuksia pieninä kokonaisuuksina, kuten myös aiemmin tehtyjä ominaisuuksia muutettaessa. Pyrin testaamaan niitä riittävästi ennen siirtymistä seuraavien ominaisuuksien kehittämiseen. Testasin ominaisuuksia ja korjauksia itsekseni yleensä useita kertoja päivässä ja lähes päivittäin työn ohjaajan kanssa. Ulkopuolinen näkökulma auttaa myös näkemään virheet ja kehityskohteet, joita ei yksin tule huomanneeksi. Puhelimella testaamista varten jakelin sovelluksen uuden version testipalvelimelle. Sovelluksen julkaisemisen apuna on Webpack, joka kokoaa kaikki sovelluksessa tarvittavat tiedostot ja riippuvuudet yhteen pakettiin. Juuritiedosto Index.html ja Webpackin luoma bundle-tiedosto täytyy vain viedä palvelimelle ja sovelluksen uutta versiota pääsee käyttämään. Pyrin pitämään sovelluksen päivittäin pääosin toimintakunnossa, jotta sitä voitiin yleensäkin testata ja tehdä tarvittaessa tarkempaa jatkomääritystä. Samoin tavoitteena oli pitää toimintakuntoinen versio sovelluksesta versionhallinnassa. On myös tärkeää tallentaa muutokset versionhallintaan lähes päivittäin, jotta kehityksessä ei tarvitse palata taaksepäin esimerkiksi laiterikon tai inhimillisen virheen takia. (Braimbridge, Ewald, Fenner., Grisogono, Hlushko, Larkin, Kaper & Stewart.)

Testaaminen tapahtui verkkoselaimella ja suurin osa testaamisesta oli käyttöliittymän toiminnan testausta. Activiti-prosessimoottori toimi paikallisessa kehitysympäristössä muistinvaraisen H2-tietokannan kanssa. Testaussession päätyttyä tietokanta poistetaan, jolloin testidataa ei tarvitse erikseen siivota tietokannasta ja toisaalta testaaminen voidaan aloittaa aina niin sanotusti puhtaalta pöydältä. Tietokanta luodaan muistiin sovelluksen käynnistyessä ja tuhotaan testauksen loppuessa. (Wicht 2010.)



KUVA 21. React Developer Tools

Testaamiseen käytin selaimista Mozillan Firefoxia, mutta myös Googlen Chromea. Puhelimessa testauskäytössä oli myös Firefoxin ja Chromen Android-versiot. Testauksen apuna oli Firefoxin ja Chromen kehittäjätila, joka näyttää tietoja sivuston toiminnasta. Esimerkiksi rajapintakutsujen onnistumista voitiin seurata tällä tavoin. Lisäksi käytössäni oli Firefoxissa kehittäjätilaan integroituva React Developer Tools-liitännäinen (KUVA 21), jolla seurasin tilojen muutoksia React-koodissa testauksen aikana. Rajapintakutsuja testasin Firefoxin RESTClient-liitännäisellä (KUVA 22) ja samantyyppisellä Postman-sovelluksella, joilla voidaan tehdä HTTP-kutsuja REST-rajapintaan. Projektin loppuvaiheessa järjestettiin pienimuotoinen sovelluksen esittely- ja testaustapahtuma kiinnostuneille yrityksen työntekijöille, joilta saatua palautetta voidaan käyttää jatkokehityksessä.



KUVA 22. Tehtävän kuittaus RESTClientillä

7 YHTEENVETO JA POHDINTA

Projektin lopputuloksena syntyvä sovellus digitalisoi, visualisoi ja dokumentoi uuden työntekijän kannalta olennaista perehdytysprosessia. Paljon käsityötä vaativa prosessin hallinta siirtyy osin sovelluksen huoleksi. Tehtävät ja niiden status näkyvät nopeasti kaaviosta ja samalla tehtävät tulevat todennäköisemmin suoritetuksi kuin unohdetuksi. Sovellus toimii riittävän luotettavasti ja täyttää vaatimukset, joten se otetaan ainakin toistaiseksi käyttöön yrityksen sisällä.

Reactin perusasiat ja työnkulku tulivat mielestäni pääosin tutuiksi. Projektin aikana opin tekemään toimivan React-käyttöliittymän. Käyttöliittymän ja käytettävyyden suunnittelu olivat innostavia ja mielestäni projektissa onnistui vaatimusten mukaisen riittävän helppokäyttöisen sovelluksen toteutus.

Reactin toiminnan ymmärtäminen oli aloittelevalle haastavaa, kuten myös dokumentaation tulkinta, ja huomasinkin, että kannattaa yrittää etsiä ohjeita ja oppaita useasta lähteestä. React-osuus oli haastava myös siksi, että hallittavia tekniikoita tarvitaan useita HTML:stä ja JavaScriptistä erilaisiin JavaScript-kirjastoihin. Toisaalta Activitin REST-rajapinta oli mielestäni helpohko omaksua ja käyttää, sillä se oli riittävän yksityiskohtaisesti dokumentoitu. Tässä todennäköisesti auttoi myös aiempi kokemukseni yksinkertaisen rajapinnan suunnittelusta ja käytöstä PHP:llä.

LÄHTEET

About Activiti. Saatavissa: https://www.activiti.org/about Viitattu: 25.3.2018.

Braimbridge, A., Ewald, J., Fenner, J., Grisogono G., Hlushko, E., Larkin, S., Kaper, R. & Stewart, J. Concepts. Saatavissa: https://webpack.js.org/concepts/. Viitattu: 5.4.2018.

Buna, S. 2017. Yes, React is taking over front-end development. The question is why. Saatavissa: https://medium.freecodecamp.org/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76. Viitattu: 3.4.2018.

Chang, R. 2016. Learning How to Build a Web Application. Saatavissa: https://medium.com/@rchang/learning-how-to-build-a-web-application-c5499bd15c8f. Viitattu: 2.4.2018.

Components – Buttons. Saatavissa: https://react-bootstrap.github.io/components/buttons/. Viitattu: 2.4.2018.

Components – Images. Saatavissa: https://react-bootstrap.github.io/components/images/. Viitattu: 2.4.2018.

Components – Panels. Saatavissa: https://react-bootstrap.github.io/components/panel/. Viitattu: 2.4.2018.

Dabit, N. 2016. Beginner's Guide to React Router. Saatavissa: https://medium.freecodecamp.org/beginner-s-guide-to-react-router-53094349669. Viitattu: 2.4.2018.

Dawson, C. 2014. JavaScript's History and How it Led To ReactJS. Saatavissa: https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/ Viitattu: 25.1.2018.

Doyle, M. 2012. Ajax with jQuery: A Beginner's Guide. Saatavissa: https://www.elated.com/articles/ajax-with-jquery-a-beginners-guide/. Viitattu: 6.5.2018.

Getting started – Introduction. Saatavissa: https://react-bootstrap.github.io/getting-started/introduction/. Viitattu: 2.4.2018.

Kirchmer, M. 2017. High Performance Through Business Process Management: Strategy Execution in a Digital World. Springer.

Krug, S. 2014. Don't Make Me Think Revisited A Common Approach To Web Usability. New Riders.

Laliwala, Z & Mansuri, I. 2014. Activiti 5.x Business Process Management Beginner's Guide. Packt Publishing.

Layout – Grid system. Saatavissa: https://react-bootstrap.github.io/layout/grid/. Viitattu: 2.4.2018.

Ludosky, S. 2015. Bootstrap Tutorial: A Guide for Beginners. Saatavissa: https://blog.udemy.com/bootstrap-tutorial-a-guide-for-beginners/. Viitattu: 26.4.2018.

Linkify-react. Saatavissa: https://soapbox.github.io/linkifyjs/docs/linkify-react.html. Viitattu: 2.4.2018.

Rascia, T. 2017. An Introduction to jQuery. Saatavissa: https://www.digitalocean.com/community/tutorials/an-introduction-to-jquery. Viitattu: 6.5.2018.

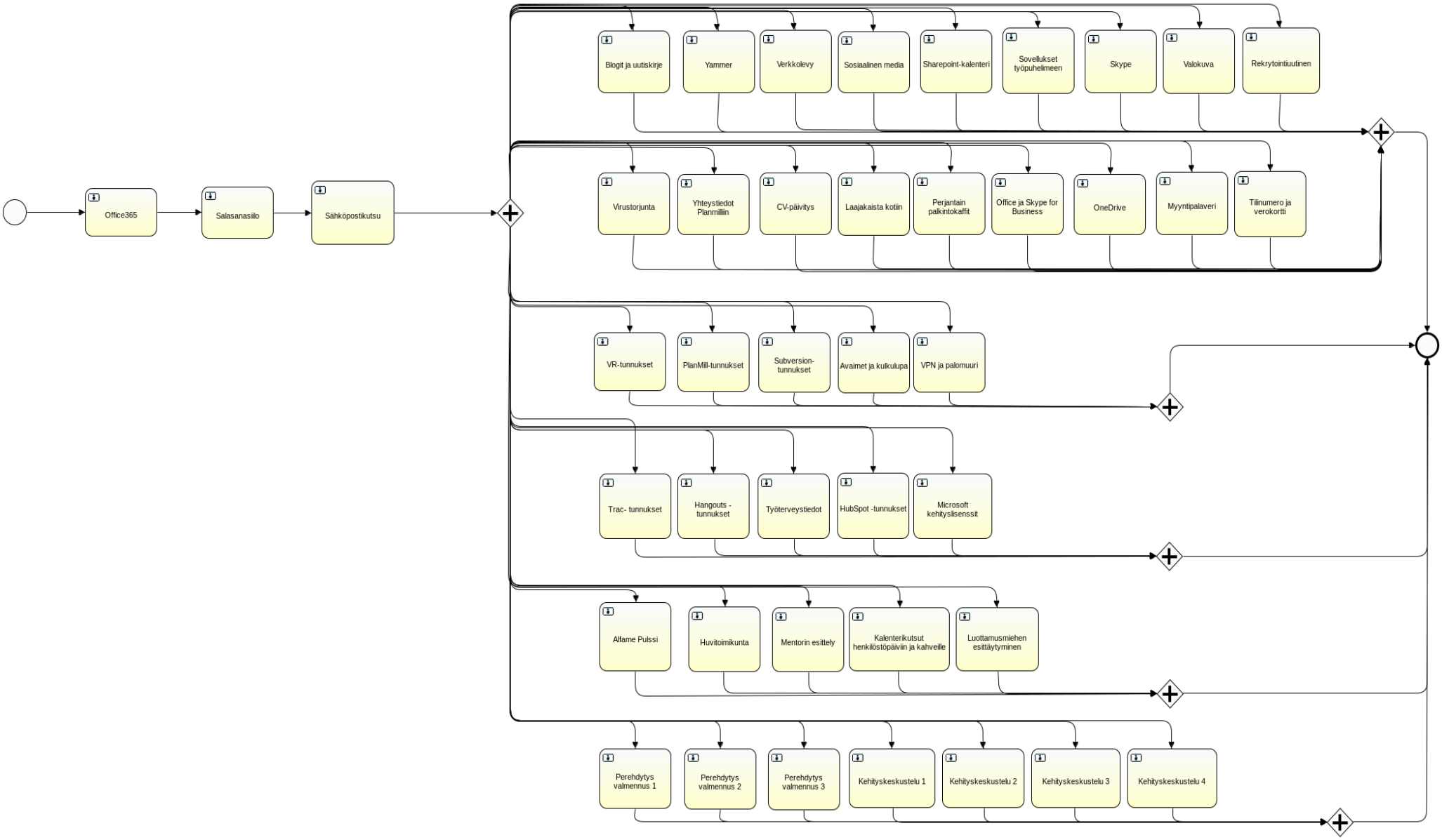
React.Component. Saatavissa: https://reactjs.org/docs/react-component.html. Viitattu: 25.3.2018.

Tenenhaus, P. 2010. Javascript:Display an image downloaded with XMLHTTPRequest. Saatavissa: http://www.philten.com/us-xmlhttprequest-image/. Viitattu: 5.4.2018.

Tero, P. 2014. The Mystery Of The jQuery Object: A Basic Introduction. Saatavissa: https://www.smashingmagazine.com/2014/05/mystery-jquery-object-syntax-basic-introduction/. Viitattu: 6.5.2018.

Tierney, C. 2017. An Absolute Beginner's Guide to Using npm. Saatavissa: https://dzone.com/articles/an-absolute-beginners-guide-to-using-npm-1. Viitattu: 26.4.2018.

Wicht, B. 2010. Presentation and Use of H2 Database Engine. Saatavissa: https://dzone.com/articles/presentation-and-use-h2. Viitattu: 21.4.2018.


Office365
Salasanasiilo
Sähköpostikutsu
Blogit ja uutiskirje
Yammer
Verkkolevy
Sosiaalinen media
Sharepoint-kalenteri
Sovellukset työpuhelimeen
Skype
Valokuva
Rekrytointiuutinen
Virustorjunta
Yhteystiedot Planmillin
CV-päivitys
Laajakaista kotiin
Perjantain palkintokaffit
Office ja Skype for Business
OneDrive
Myyntipalaveri
Tilinumero ja verokortti
VR-tunnukset
PlanMill-tunnukset
Subversion-tunnukset
Avaimet ja kulkulupa
VPN ja palomuuri
Trac- tunnukset
Hangouts -tunnukset
Työterveystiedot
HubSpot -tunnukset
Microsoft kehityslisenssit
Alfame Pulssi
Huvitoimikunta
Mentorin esittely
Kalenterikutsut henkilöstöpäiviin ja kahveille
Luottamusmiehen esittäytyminen
Perehdytys valmennus 1
Perehdytys valmennus 2
Perehdytys valmennus 3
Kehityskeskustelu 1
Kehityskeskustelu 2
Kehityskeskustelu 3
Kehityskeskustelu 4